

# VCS Quick Start

[Get The PDF Version Here](#)

## Notations

s represent a slab/transient variable

s values are assumed going from 230 to 310

x is the vcs canvas object, i.e.

```
x=vcs.init)
```

## Starting up

```
import vcs # import the vcs module
x=vcs.init() # initiate the vcs window
x.plot(s) # Plot the slab s
```

to change the orientation:

```
x.open() # pops up the vcs window
x.page() # switch between portrait and landscape mode
x.portrait() # Set the window to portrait mode
x.landscape() # set the window to landscape mode
x.geometry(widht,height,xoff,yoff) # to specify your own window size
```

## Defining VCS Objects and using them

Available graphic methods are:

**'boxfill', 'isofill', 'isoline', 'meshfill', 'taylordiagram', 'vector', 'xvsv', 'outfill', 'outline', 'scatter', 'xyvsv', 'yxvsv', 'continents'**

to list the methods available for a certain type (say 'isofill')

```
x.show('isofill')
```

to retrieve a boxfill method 'quick':

```
iso=x.getisofill('quick')
```

Or to create a new isofill method named 'new' from 'quick':

```
iso=x.createisofill('new',['quick']) #'quick' is optional if not passed
then the 'default' is used
```

you can now plot s using "iso":

```
x.plot(s,iso)
```

an other type of VCS objects is 'template, i.e. where things are plotted in the vcs window:

```
tmpl=x.gettemplate('quick')
#or
```

```
tmpl=x.createtemplate('new',['quick'])
```

now you can plot s with iso in the template tmpl:

```
x.plot(s,iso,tmpl)
```

other vcs objects are used, they represent specifications used by graphic-methods or template, these object are:

**'line', 'marker', 'fillarea', 'textorientation' and 'textable'**

Once again you can read them in with the get or create new ones with create, ex:

```
fa=x.createfillarea('new')
```

## Altering VCS Objects

a list of the modifiable attributes of any vcs object can be obtain by using its list() function:

```
iso.list()
```

the quick reference shows the available attributes for each vcs objects

cf the following section (examples) for more specific instructions

## Outputing

### Image

to the screen:

```
x.plot(s,[method],[template])
```

to the printer:

```
x.printer('printername')
```

to a file:

postscript:

```
x.postscript('filename[.ps]',[orientation]) # orientation is 'l'/'p' autodetected
```

gif:

```
x.gif('filename[.gif]',[append],[orientation]) #append: 'a'/'r'
```

cgm:

```
x.cgm('filename[.cgm]',[append])
```

raster:

```
x.raster('filename[.ras]',[append])
```

## VCS object (obj)

A vcs object can be saved as either a python script or a vcs script by using its script() function: iso.script('filename'), vcs script will be saved is 'filename' ends by .scr

## Examples

### Usefull method available to create levels/colors/palette

The VCS module comes with some handy functionalities:

```
#returns the max and min
max,min=vcs.mxmn(slab or list of slabs)

levs=vcs.mkevenlevels(n1,n2,[n])
# returns a list of levels going from n1 to n2 and spanning n intervals
#(i.e. levs has n+1 values), the default value for n is 10 intervals

levs=vcs.mkscale(n1,n2,[nmax],[zero])
#return a list of levels containing n1 and n2, with a maximum of nmax values.
#Zero determines if zero can be a level (zero=1, default), has to be a level
(zero=2) or cannot be a level (zero=-1)

cols=vcs.getcolors(levs,[cols=range(16,40)],[white=240],[split=1]),
#returns a set of colors spanning the list "cols". If
#"split" is set to 1 (default) the "cols" list is split in 2
#halves one for >0 values, 1 for <0 values, moreover if a contour goes
#from <0 to >0 it will be set to the color "white" (default=241,
#i.e. White)

lbls=vcs.mklabls(levs,output='dic')
#return a dictionary containing "nicely" rewritten valus for the
#levels ex: 0.00000001 -> '1E-8', if output is not starting with 'dic' then a
#simple list is returned
```

### Playing with the levels/colors/attributes

Let's assume we want to redefine the levels for displaying s, say from 250 up to 300, every 5, with extension arrows. Here is how to do it for different graphic methods:

Thereafter the levels wanted are in the "levs" list and the labels dictionary in "lbls" and the colors are in "cols"

### Boxfil

```
box=x.createboxfill('new')
box.level_1=levs[0]
box.level_2=levs[1]
box.ext_1='y'
box.ext_2='y'
box.legend=lbls
x.plot(s,box)
```

## Isofill

```
iso=x.createisofill('new')
levs.append(1.E20)          # to have an extension arrow on the left
levs.insert(0,1.E20)       # to have an extension arrow on the right
iso.levels=levs            # sets the levels
iso.fillareacolors=cols    # set the colors
x.plot(s,iso)
```

## Isoline

```
iso=x.createisoline('new')
levs.append(1.E20)
levs.insert(0,1.E20)
iso.level=levs
iso.line='dash',
or list of line types
x.plot(s,iso)
```

yvsx (yxvsx or xyvsy), 2D plot (y(x))

```
mx,mn=x.mxm(n,y)
levs=x.mkscale(mx,mn)
xy=x.createxvsy('new')
xy.datawc_y1=levs[0]
xy.datawc_y2=levs[-1]
xy.datawx_x1=x[0]
xy.datawc_x2=x[-1]
x.plot(x,y,xy)
```

## Altering the Template

To create a vcs template object:

```
tmpl=x.createtemplate('new')
```

The data area is in: `tmpl.data` and is represented by the 4 edges `x1<x2` and `y1<y2`, the values are in % of the canvas. For example to plot your data for 25 to 75 % of the page horizontally and from 30% to 70% vertically:

```
tmpl.data.x1=.25 ; tmpl.data.x2=.75 ; tmpl.data.y1=.3 ; tmpl.data.y2=.7
```

The box around the data is determined by `box1`, therefore we should do:

```
tmpl.box1.x1=.25 ; tmpl.box1.x2=.75 ; tmpl.box1.y1=.3 ; tmpl.box1.y2=.7
```

Finally the horizontal tickmarks are set by `xtickmark` and `xtic1` (`xtic2` for the second ones), and the labels by `xlabel1` (2)

```
tmpl.xtick1.y1=.29;tmpl.xtic1.y2=.3;tmpl.xlabel1.y=.27
```

Setting the vertical labels works the same way:

```
tmpl.ytick1.x1=.24;tmpl.ytic1.x2=.25;tmpl.ylabel1.x=.22
```

Note the actual values of the labels are set in the vcs graphics method, for example on isofill (but it works the same on ANY graphic method)

```
iso.xticlabels1=dictionary of location/names
```

Finally any attributes of the template can be set on/off using the priority attribute, for example to turn the legend off:

```
tmpl.legend.priority=0
```

Remember a list of the element of the templates that you can set is accessible using `tmpl.list()` function. Also if you know which element you wish to set a less exhaustive can be obtained by using the list function of this element for example: `tmpl.data.list()` will the attribute of "data" zone that you can set.